Spirit Clash
Token Audit

The BlockChain Auditor

Prepared by: Jorge Martinez

Version: 1
Date: Jan. 23, 2020

Summary of Findings

This document expresses all security concerns of the Spirit Clash Smart Contracts as expressed by Jorge Martinez. I took care to attempt to find as many ways to improve the security, code efficiency, best practices, and overall function of the smart contracts.

Contract Status: Safe to Use

0 Critical Issues found were found.
0 Medium Issues were found.
4 Low Issues in the governance.sol were found.
0 Informational Issues were found.

Solidity Code Coverage

Jorge's Test Suite	Spirit Clash	Industry Standard
100%	0%	95%

For this audit, I wasn't provided with a testing suite but as part of my audit methodology I developed a test suite to verify the functionality of the ERC20 contract, check its security, and to help reveal any underlying issues.

This audit should be seen as one step in the development process with the intent of raising awareness on the meticulous work involved in secure development and making no material statements or guarantees to the operational state of the smart contract(s) once they are deployed. This document is not an endorsement of the reliability or effectiveness of the smart contracts. This is an assessment of the smart contract logic, implementation, and best practices. I cannot take responsibility for any potential consequences of the deployment or use of the smart contract(s) related to the audit.

Test Suite Results

Jorge’s Test Suite

```
Spirit Clash Token Test Suite
  Token Functionality
    deployment
      ✓ set name to Clash Token (491ms)
      ✓ sets symbol to SCT
      ✓ has 18 decimals
      ✓ owner is the deployer
      ✓ treasury address set
      ✓ treasury owns all the tokens initially (40ms)
    fallback
      ✓ fallback reverts if eth is sent to it (51ms)
    allowance
      ✓ it returns an address' allowance (220ms)
    approve
      ✓ cannot approve the zero address
      ✓ msg.sender cannot approve themself
      ✓ approve allows an address to approve another address to transfer their tokens (100ms)
    decreaseApproval
      ✓ can decrease another address' allowance (68ms)
      ✓ can't decrease the zero addresses allowance
      ✓ an address cannot decrease their own allowance
      ✓ if decreasing an address' allowance will overflow it is set to zero (44ms)
    increaseApproval
      ✓ cannot increase the allowance of the zero address
      ✓ an address cannot increase its own allowance
      ✓ increasing an address' allowance can overflow but will never decrease below its current value (57ms)
      ✓ an address can increase another address' allowance
    transfer
      ✓ cannot transfer tokens to yourself
      ✓ cannot transfer to the zero address
      ✓ cannot transfer tokens the Clash Token contract
      ✓ reverts if more tokens are transferred than owned
      ✓ untestable: can make another address' balance overflow but if it decreases the receivers balance it will revert
      ✓ unreachable: reverts if you try to transfer more tokens than you own
      ✓ holders can transfer tokens (54ms)
    transferFrom
      ✓ untestable: cannot transfer tokens from the zero address
      ✓ untestable: transferFrom reverts if tokens are sent out from Spirit Clash
      ✓ reverts if you send tokens from one address to itself (60ms)
      ✓ reverts you transfer an address' tokens to the zero address (42ms)
      ✓ transferFrom reverts if it is used to send tokens to Spirit Clash (40ms)
      ✓ cannot transfer more tokens out of address than it owns (42ms)
      ✓ cannot transfer more tokens out of an address than allowed (39ms)
    transferOwnership
      ✓ only the owner can call this function
      ✓ new owner cannot be the zero address
      ✓ cannot make ClashToken contract address the owner
      ✓ new owner cannot be the current owner
      ✓ Spirit Clash is able to change owners


  38 passing (2s)

----------------|----------|----------|----------|----------|----------------|
File            |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------|----------|----------|----------|----------|----------------|
 contracts/     |      100 |    88.89 |      100 |      100 |                |
  ClashToken.sol|      100 |    88.89 |      100 |      100 |                |
----------------|----------|----------|----------|----------|----------------|
All files       |      100 |    88.89 |      100 |      100 |                |
----------------|----------|----------|----------|----------|----------------|
```

Table of Contents

Audit Methodology & Techniques

The BlockChain Auditor has the following auditing process:

1. Our audits include
   a. Review of the specifications, source code, and instructions provided to the TheBlockchainAuditor to clearly identify the desired functionality of the smart contract(s).
   b. Manual line by line review of contract code to spot potential vulnerabilities.
   c. Identification of deviations between desired functionality expressed to the TheBlockchainAuditor and what the smart contract(s) are doing.
2. Automated static and symbolic analysis, as well as verifying testing coverage using the provided test suite.
   a. Automated static and symbolic analysis help determine what inputs cause each part of the smart contract to execute. Analysis of how much of the code base is tested and comparison to industry standard.
3. Examination of smart contracts and development process as a whole, ensuring best practices are followed, allowing improved efficiency and security based on established industry and academic practices.
4. Specific, itemized, and actionable recommendations to assist in securing the smart contract(s) in question.

Contract Checklist

ClashToken.sol

Contract Vulnerability	
Integer Overflow	Pass*
Race Condition	Pass
Denial of Service	Pass
Logical Vulnerability	Pass
Hardcoded Address	Pass
Function Input Parameter Check	Pass
Function Access Control Check	Pass
Random Number Generation	N/A
Random Number Use	N/A
Contract Specification	
Solidity Compiler Version	Pass
Event Use	Pass
Fallback Function Use	Pass
Constructor Use	Pass
Function Visibility Declaration	Pass
Variable Storage Declaration	Pass
Deprecated Keyword Use	Pass
ERC20/223 Standard	N/A
ERC721 Standard	N/A
Business Risk	
Able to Arbitratrily Create Token	N/A
Able to Arbitrarily Destroy Token	N/A
Can Suspend Transactions	Fail
Short Address Attack	Pass
Gas Optimization	
assert()/require()/revert() misused	Pass
Loop Optimization	Pass
Storage Optimization	Pass

Issue Classification

Critical

These issues in the smart contract can have catastrophic implications that could ruin your reputation, disrupt the contract's functionality, or impact the client and your users' sensitive information.

Informational

This issue relates to style and security best practices but does not pose an immediate risk.

Medium

An issue classified as medium has relatively small risk and isn't exploitable to circumvent desired functionality and could not have financial consequences but could put user's sensitive information at risk.

Acknowledged

The issue remains in the code but is a result of an intentional business or design decision.

Low

An issue classified as informational does not pose an immediate threat to disruption of functionality and could not be exploited on a recurring basis, however, it should be considered for security best practices or code integrity.

Unresolved

Although the client has been informed of the risk, it was decided to accept it because it was not relevant in the functionality of the smart contract.

Undetermined

The impact of the issue is uncertain and more investigation is required to understand the repercussions of the issue.

Mitigated

Actions were taken to minimize the impact or likelihood of the risk.

Detailed Analysis

5.1 transfer()

ClashToken.sol 109

Redundant require() statement

Explanation:

require(_value <= transferableTokens(msg.sender)) on line 109 is actually the same thing as **require(balances[msg.sender] - _value <= balances[msg.sender])** on line 107.

Recommendation:

Since we are already verifying that the sender has enough tokens on line 107 it would have been okay to remove the require() statement on line 109.

5.2 transfer()

ClashToken.sol 108

Redundant require() statement

Explanation:

require(balances[_to] <= balances[_to] + _value) on line 108 is making sure that not amount send to an address will ever be so high that it causes the receiver's address to overflow and end up as a lower value. Unless there is some exploit to generate more tokens it would be impossible for this to happen and given the token supply I am unable to test this without increasing the total supply.

Recommendation:

I think that this is a good require() statement to have but it seems very unlikely that it will ever be used and this statement could be removed. As it is, it is one more layer of security.

5.3 transferFrom()

ClashToken.sol 120

Redundant require() statement

Explanation:

require(_from != address(0)) on line 120 prevents an address to transfer tokens from the zero address to another address. This is unnecessary because **transfer()** doesn't allow you to send tokens to the zero address and its likely no one will ever guess the private key of the zero address.

Recommendation:

You could remove this require() statement but it does add an additional layer of security, albeit seemingly uncenecessary.

5.4 transferFrom()

ClashToken.sol 121

Redundant require() statement

Explanation:

require(_from != address(this)) on line 121 is unnecessary because **approve()** won't let you set yourself an allowance.

Recommendation:

You could remove this require() statement but it does add an additional layer of security, albeit seemingly uncenecessary.

5.5 transferFrom()

ClashToken.sol 127

Redundant require() statement

Explanation:

require(balances[_from] - _value <= balances[_from]) on line 127 is not necessary because on line 125 **require(_value <= transferableTokens(_from))** is already checking that the from address has enough enough tokens.

Recommendation:

Since we are already verifying that the sender has enough tokens on line 125 it would have been okay to remove the require() statement on line 127.

5.6 transferFrom()

ClashToken.sol 128

Redundant require() statement

Explanation:

require(balances[_to] <= balances[_to] + _value) on line 128 given the token supply there is no way that a single account could ever own enough tokens for a transfer to cause another account's balance to overflow which is what this require statement was ultimately written to do.

Recommendation:

I think that this is a good require() statement to have but it seems very unlikely that it will ever be used and this statement could be removed. As it is, it is one more layer of security.

5.7 transferOwnership()

ClashToken.sol 139

Unclear Use

Explanation:

Although the ownership is functional and you will be able to transfer ownership to another address or contract, I don't really see what is gained from having this as there are no functions that have their access limited.

Recommendation:

If you were to redeploy I would remove this function. As it is it neither hurts nor helps this token.

5.8 transferableTokens()

ClashToken.sol 151

Unclear Use

Explanation:

I don't see the point of transferableTokens(), it is essentially just a wrapper around balanceOf() that doesn't add anything to it.

Recommendation:

If you were redeploy I would remove this function. As it is it neither helps no hurts this token.

Executive Summary

Overall Thoughts

During my analysis I wrote an extensive test suite to see the functionality of the Spirit Clash Token contract. Initially, my biggest concern was that this contract was not using OpenZeppelin's SafeMath library; however, my test suite found that the require statements that were used in safeguard transfer() and transferFrom() calls sufficient increasing and decreasing users' balances. It performs all the functionality that is necessary as an ERC20 token and correctly implements EIP20. The only issues I found were simply informational and I was not able to find any exploits. As I was not able to find any critical, medium, or low issues I feel comfortable recommending this token to be used on mainnet for the Spirit Clash protocol.

Appendix A

File Fingerprints

ClashToken.sol	814456cfea81f7da5584093802a77caa

The Blockchain Auditor is honored to have the opportunity to help provide enhanced security and efficiency for the Spirit Clash platform.

I look forward to seeing how Spirit Clash will integrate the Spirit Clash Token in their game platform.

The BlockChain Auditor

- Jorge Martinez